Advanced Java Lab

Week-12

Roll Number: 238W1A12C4

1. Integrate MySQL with Spring Boot & Perform Operations

Code:

application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/springdb
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

Book.java:

```
package com.example.demo;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class Book {

    @Id
    private int id;
    private String title;

    public Book() {}
    public Book(int id, String title) {
        this.id = id;
        this.title = title;
    }

    // Getters & Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
}
```

BookRepository.java:

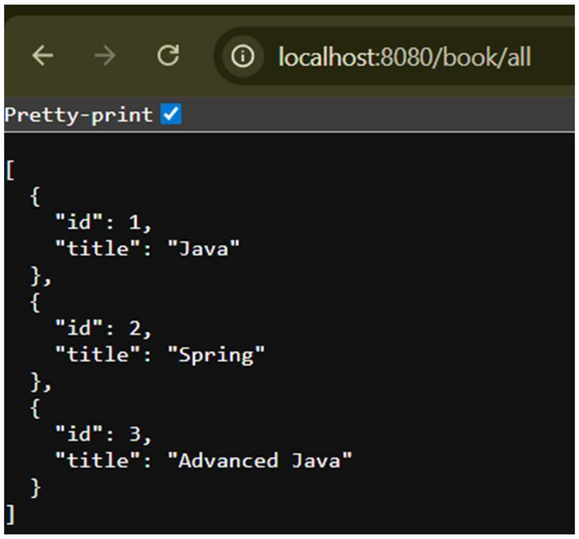
```
package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Integer> {
}
```

BookController.java:

```
package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
```

```
@RestController
@RequestMapping("/book")
public class BookController {

    @Autowired
    private BookRepository repo;

    // INSERT → http://localhost:8080/book/add?id=1&title=Java
    @GetMapping("/add")
    public String addBook(@RequestParam int id, @RequestParam String title) {
        Book b = new Book(id, title);
        repo.save(b);
        return "Book Added Successfully!";
    }

    // GET ALL → http://localhost:8080/book/all
    @GetMapping("/all")
    public List<Book> getAllBooks() {
        return repo.findAll();
    }

    // GET BY ID → http://localhost:8080/book/get?id=1
    @GetMapping("/get")
    public Book getBook(@RequestParam int id) {
        return repo.findById(id).orElse(null);
    }

    // DELETE → http://localhost:8080/book/delete?id=1
    @GetMapping("/delete")
    public String deleteBook(@RequestParam int id) {
        repo.deleteById(id);
        return "Book Deleted Successfully!";
    }
}
```

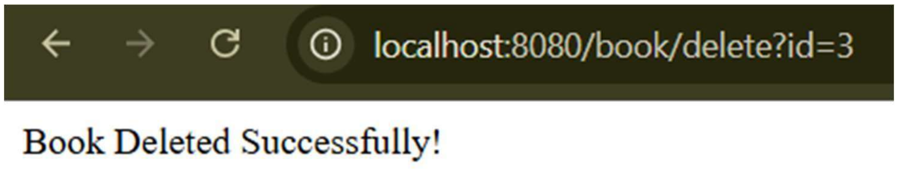
Output:



localhost:8080/book/all

Pretty-print ✓

```
[
  {
    "id": 1,
    "title": "Java"
  },
  {
    "id": 2,
    "title": "Spring"
  },
  {
    "id": 3,
    "title": "Advanced Java"
  }
]
```



localhost:8080/book/delete?id=3

Book Deleted Successfully!

2. Write custom query methods using Spring Data JPA method naming conventions.

Code:

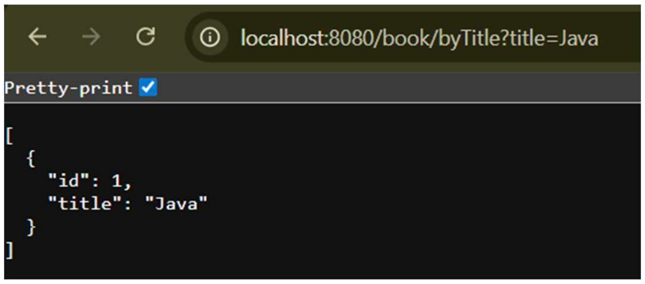
BookRepository.java:

```
package com.example.demo;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface BookRepository extends JpaRepository<Book, Integer> {
    List<Book> findByTitle(String title);
    List<Book>  findByTitleContaining(String  keyword);


    List<Book>  findByIdGreaterThan(int  id);


}
```

BookController.java:

```
@GetMapping("/byTitle")
public List<Book> findByTitle(@RequestParam String title) {
    return repo.findByTitle(title);
}
@GetMapping("/search")
public List<Book> search(@RequestParam String keyword) {
    return repo.findByTitleContaining(keyword);
}
@GetMapping("/greater")
public List<Book> findGreater(@RequestParam int id) {
    return repo.findByIdGreaterThan(id);
}
```

Output:



```
[
  {
    "id": 1,
    "title": "Java"
  }
]
```